REMARKS/ARGUMENTS

Claims 1-13 are pending in the present application. Claims 1, 5, and 10 are amended to address the §112(1) rejection. Reconsideration of the claims is respectfully requested.

I. 35 U.S.C. §112, First Paragraph

Claims 1 and 10 stand rejected under 35 U.S.C. §112, first paragraph as failing to comply with the written description requirements for reciting a "mountable device". In response, these claims are amended to delete the word "mountable." Additionally, the same amendments are made to claim 5, which has the same phrase and would carry the same rejection. Therefore, the rejection of these claims under 35 U.S.C. §112, first paragraph has been overcome.

II. 35 U.S.C. §102, Anticipation

Claims 1-7 and 9-13 are rejected under 35 U.S.C. §102 as anticipated by *Kidder, et al.*, Configurable Fault Recovery Policy for a Computer System, U.S. Patent 6,983,362 (January 3, 2006) (hereinafter *Kidder*). This rejection is respectfully traversed. The rejection states:

> Regarding claim 1, Kidder et al. discloses a computer system and a computer program product on a computer readable medium, comprising: a first processor (24) connected as a server; a plurality of client processors (26a,b,c,etc.) connected to communicate with said first processor (see fig. 1); a filesystem stored on at least one mountable device (see column 12, line 50+) and connected for accessed from said first processor (24) and said plurality of client processors (26); and a set of instructions configured to run on said computer system, wherein when a first portion of said filesystem is found to be corrupt (see column 35, line 41+), said first portion being on a first mountable device of said at least one mountable device (see column 39, line 5+ and see figs. 25-26), said set of instruction are connected to: receive information regarding a location of paid first portion and a perceived corruption (see column 35, line 40+), in response to receipt of said information. (See column 6, line 10-17), isolate said first portion of said filesystem while leaving other portions of said filesystem available (see column 35, line 27+), while said first portion of said filesystem is isolated (See column 35, line 27+), while said first portion of said filesystem is isolated (See column 35, line 27+), and provide repair for said filesystem and after repair of said first portion of said file system (see column 40, line 27+), remove the isolation of said first portion (in order to restarting a single software, the isolation have to remove). (See column 38, line 60).

Final Office Action dated October 23, 2006, pp. 3-4.

Claim 1, as amended states:

> 1. A computer system, comprising:
> a first processor connected as a server;
> a plurality of client processors connected to communicate with said first processor;
> a filesystem stored on at least one device and connected for access by said first processor and said plurality of client processors; and
> a set of instructions configured to run on said computer system, wherein when a first portion of said filesystem is found to be corrupt, said first portion being on a first device of said at least one device, said set of instructions are connected to:
> receive information regarding a location of said first portion and a perceived corruption,
> in response to receipt of said information, isolate said first portion of said filesystem while leaving other portions of said filesystem available,
> while said first portion of said filesystem is isolated, provide repair for said first portion of said filesystem; and
> after repair of said first portion of said file system, remove the isolation of said first portion.

Kidder does not anticipate claim 1 because this reference does not disclose a number of features recited in this claim. *Kidder* is not directed to the detection and repair of corruption in a filesystem, as recited in claim 1, and this reference does not disclose instructions that perform the recited steps in response to "*a first portion of said filesystem [being] found to be corrupt*". Additionally, *Kidder* does not disclose instructions that are connected to "*isolate said first portion of said filesystem while leaving other portions of said filesystem available*", to "*provide repair for said first portion of said filesystem*", or to "remove the isolation of said first portion."

The examiner asserts otherwise, citing the following portion of *Kidder*:

> Referring again to FIG. 1, on power-up, reset or reboot, the processor on each board (central processor and each line card) downloads and executes boot-strap code (i.e., minimal instances of the kernel software) and power-up diagnostic test code from its local memory subsystem. After passing the power-up tests, processor 24 on central processor 12 then downloads kernel software 20 from persistent storage 21 into non-persistent memory in memory subsystem 28. Kernel software 20 includes operating system (OS), system services (SS) and modular system services (MSS).

Kidder, column 12, lines 50-60.

Kidder is directed towards fault and event management being carried out according to a configurable fault recovery policy. This portion of *Kidder* teaches the booting process for a computer, including putting kernel software into memory. The examiner appears to equate the claimed filesystem to the kernel software of *Kidder*, which is part of the operating system and which is downloaded from

persistent memory. However, one of ordinary skill in the art would not consider the kernel software to be equivalent to a filesystem. The kernel software is that portion of the operating system that is loaded first and that remains in system memory throughout execution of the operating system, while a filesystem is simply a collection of files. A kernel is able to interface a filesystem, but a kernel is not a filesystem. Therefore, this section of *Kidder* does not disclose the feature "*a filesystem stored on at least one device and connected for access by said first processor and said plurality of client processors*" as recited in claim 1.

Further, assuming *arguendo* that this inference should be made, if the Examiner believes that the system kernel should be read on the claimed filesystem, then this interpretation should be used consistently throughout the rejection. This interpretation is not consistently used, as is pointed out more specifically below.

> Traditionally, fault detection and monitoring does not receive a great deal of attention from network equipment designers. Hardware components are subjected to a suite of diagnostic tests when the system powers up. After that, the only way to detect a hardware failure is to watch for a red light on a board or wait for a software component to fail when it attempts to use the faulty hardware. Software monitoring is also reactive. When a program fails, the operating system usually detects the failure and records minimal debug information.
>
> Current methods provide only sporadic coverage for a narrow set of hard faults. Many subtler failures and events often go undetected. For example, hardware components sometimes suffer a minor deterioration in functionality, and changing network conditions stress the software in ways that were never expected by the designers. At times, the software may be equipped with the appropriate instrumentation to detect these problems before they become hard failures, but even then, network operators are responsible for manually detecting and repairing the conditions.
>
> Systems with high availability goals must adopt a more proactive approach to fault and event monitoring. In order to provide comprehensive fault and event detection, different hierarchical levels of fault/event management software are provided that intelligently monitor hardware and software and proactively take action in accordance with a defined fault policy. A fault policy based on hierarchical scopes ensures that for each particular type of failure the most appropriate action is taken. This is important because over-reacting to a failure, for example, re-booting an entire computer system or re-starting an entire line card, may severely and unnecessarily impact service to customers not affected by the failure, and under-reacting to failures, for example, restarting only one process, may not completely resolve the fault and lead to additional, larger failures. Monitoring and proactively responding to events may also allow the computer system and network operators to address issues before they become failures. For example, additional memory may be assigned to programs or added to the computer system before a lack of memory causes a failure.

Kidder, column 35, line 42 through column 36, line 14.

This excerpt discusses the detection of faults in general terms. No specific mention of corruption in a filesystem is disclosed. Additionally, no specific mention of corruption in the kernel software, which was previously read on the filesystem by the examiner, is disclosed. Instead, *Kidder* discloses failures in software applications and in hardware. The cited section teaches preventing re-booting to many programs or computers, or to reboot enough programs or computers to correct problems. *Kidder* does not disclose in this section or elsewhere "*when a first portion of said filesystem is found to be corrupt,*" as recited in claim 1. Additionally, while the above excerpt mentions responding to faults by giving programs additional memory access, no mention is made in this section of isolating a portion of the filesystem while leaving other portions available, as claimed.

A more relevant portion of the reference should be cited and discussed to further show that *Kidder* not only does not teach the features as arranged in claim 1, but in fact teaches away from claim 1. Specifically, *Kidder* teaches that:

> In addition, the OSE operating system includes memory management that supports a "protected memory model". The protected memory model dedicates a memory block (i.e., defined memory space) to each process and erects "walls" around each memory block to prevent access by processes outside the "wall". This prevents one process from corrupting the memory space used by another process. For example, a corrupt software memory pointer in a first process may incorrectly point to the memory space of a second processor and cause the first process to corrupt the second processor's memory space. The protected memory model prevents the first process with the corrupted memory pointer from corrupting the memory space or block assigned to the second process. As a result, if a process fails, only the memory block assigned to that process is assumed corrupted while the remaining memory space is considered uncorrupted.

Kidder, column 13, lines 35-51.

This portion of *Kidder* teaches that corruption of memory can occur, but does not treat this corruption as does the claims. *Kidder* acts before any fault or corruption has occurred by partitioning portions of the memory for each program. Other programs cannot access these memory portions and programs do not use memory from other regions. This procedure prevents a program from corrupting another program's memory. In other words, this procedure prevents faults from spreading between programs, thus minimizing the number of programs to reboot to correct a fault. Rather than repair any corruption, *Kidder* isolates the memory ahead of time so that when corruption occurs, only one process needs to be rebooted.

In contrast, the features of the claims include responding to a corruption in the filesystem. Specifically, "when a first portion of said filesystem is found to be corrupt" instructions are initiated that "isolate said first portion of said filesystem while leaving other portions of said filesystem available." Further, "while said first portion of said filesystem is isolated, provide repair for said first portion of said

filesystem." These features are completely different than the teachings of *Kidder*. *Kidder* contains no mention of detecting corruption of the filesystem and isolating the corruption, as claimed. *Kidder* also contains no mention of repair of the corruption, as claimed, as further discussed below. *Kidder* actually teaches against this claimed method by saying that repair is difficult, if not impossible, and then providing a completely different solution to the problem.

Nevertheless, the examiner also cites the following text from *Kidder*:

> The modification may cause the hierarchical scope to react more or less aggressively to particular known faults or events, and the modification may add recovery actions to handle newly learned faults or events. The modification may also provide a temporary patch while a software or hardware upgrade is developed to fix a particular error.
>
> If an application runs out of memory space, it notifies the operating system and asks for more memory. For certain applications, this is standard operating procedure. As an example, an ATM application may have set up a large number of virtual circuits and to continue setting up more, additional memory is needed. For other applications, a request for more memory indicates a memory leak error. The fault policy may require that the application be re-started causing some service disruption. It may be that re-starting the application eventually leads to the same error due to a bug in the software. In this instance, while a software upgrade to fix the bug is developed, a temporary patch to the fault policy may be necessary to allow the memory leak to continue and prevent repeated application re-starts that may escalate to line card re-start or fail-over and eventually to a re-boot of the entire computer system. A temporary patch to the default fault policy may simply allow the hierarchical scope, for example, the local resiliency manager or the slave SRM, to assign additional memory to the application. Of course, an eventual re-start of the application is likely to be required if the application's leak consumes too much memory.
>
> A temporary patch may also be needed while a hardware upgrade or fix is developed for a particular hardware fault. For instance, under the default fault policy, when a particular hardware fault occurs, the recovery policy may be to fail-over to a backup board. If the backup board includes the same hardware with the same hardware bug, for example, a particular semiconductor chip, then the same error will occur on the backup board. To prevent a repetitive fail-over while a hardware fix is developed, the temporary patch to the default fault policy may be to restart the device driver associated with the particular hardware instead of failing-over to the backup board.

Kidder, column 40, lines 27-67.

This excerpt teaches how the fault policy can be configured to avoid over-reacting to a situation while a permanent solution is implemented. No repair is performed by *Kidder*, as is claimed. *Kidder* mentions the development of software upgrades and a hardware fix, but these corrections are not

equivalent to the claimed features. This excerpt does not disclose either that repair is performed on a filesystem or that any action is taken on the kernel software that was read on the claimed filesystem. Thus, *Kidder* does not disclose "*provid[ing] repair for said first portion of said filesystem*", as recited in claim 1.

Additionally, *Kidder* teaches that:

> Restarting a single software process is much faster than switching over an entire board to a redundant board or re-booting the entire computer system. When a single process is restarted, only a fraction of a card's services are affected.

Kidder, column 38, lines 60-63.

This section of *Kidder*, taken in context, is part of determining how to react to a failure. To speed recovery, *Kidder* finds that only one application needing to be rebooted rather than switching many processes to other hardware is preferable. The isolation feature of *Kidder*, namely pre-failure isolation of memory, is not mentioned, nor implied. To show this feature, *Kidder* would have to stop isolating memory at some point. Stopping isolation would be contrary to the entire purpose of the isolation protocol implemented in *Kidder*; namely, prevention of faults spreading to multiple processes. However, the claimed feature requires that the isolated memory be released. Therefore, *Kidder* does not disclose the feature of removal of "the isolation of said first portion," as claimed.

Applicants have demonstrated that *Kidder* does not disclose numerous elements of claim 1. Because *Kidder* does not identically disclose every feature of claim 1, this reference does not anticipate claim 1. Claims 5 and 10 have been rejected for reasons that are similar to the reasons used against claim 1. Therefore, the anticipation rejection has been overcome.

Additionally, claims 2, 6, and 11 claim other additional combinations of features not suggested by the reference. Regarding claim 2, the Examiner cited *Kidder*, column 35, line 42+ (quoted above) as teaching all the features of claim 2. Claim 2 is as follows:

> 2. The computer system of Claim 1, wherein said set of instructions receives said information from a scout process that traverses the filesystem looking for corruption.

However, *Kidder* does not teach all the features of this claim. For example, the cited section does not teach looking for corruption in the filesystem, as claimed. As mentioned above, this section teaches fault correction by not over or under-reacting to a fault. *Kidder* does not seek out memory corruption, but seeks out failures in hardware or software. Action is taken with regards to these faults according to a fault policy. Nowhere in this section, or anywhere in *Kidder*, does the process seek out corruption in the

filesystem, whether the kernel is read onto the filesystem or not. As no attempt is made by *Kidder* to look for corruption in the filesystem, *Kidder* does not teach all the features of claim 2. Therefore, *Kidder* does not anticipate claim 2. By the same analysis, *Kidder* does not anticipate claims 6 and 11. Therefore, the rejection of claims 2, 6 and 11 is overcome.

Further, because claims 2-4 depend from claim 1, claims 6, 7 and 9 depend from claim 5, and claims 11-13 depend from claim 10, the distinctions between *Kidder* and the invention recited in claim 1 are equally applicable to these claims. Consequently, the rejection of claims 1-7 and 9-13 under 35 U.S.C. §102 has been overcome.

Furthermore, *Kidder* does not teach, suggest, or give any incentive to make the needed changes to reach the presently claimed invention. *Kidder* actually teaches away from the presently claimed invention because *Kidder* teaches prevention of memory corruption spreading by isolation before processes use memory opposed to isolation of corrupted filesystems as in the presently claimed invention. Further, *Kidder* emphasizes the difficulty of fixing corrupt memory, calling this repair "impossible" in many instances. The independent claims of the present invention all contain features that repair memory corruption. The solution of *Kidder* teaches away from using the solution of the presently claimed invention. Absent the Examiner pointing out some teaching or incentive to implement isolation and repair of a filesystem in *Kidder*, one of ordinary skill in the art would not be led to modify *Kidder* to reach the present invention when the reference is examined as a whole. Absent some teaching, suggestion, or incentive to modify *Kidder* in this manner, the presently claimed invention can be reached only through an improper use of hindsight using the applicants' disclosure as a template to make the necessary changes to reach the claimed invention.

III. 35 U.S.C. §103, Obviousness

Claim 8 stands rejected under 35 U.S.C. §103(a) as obvious over *Kidder*. This rejection is respectfully traversed.

Claim 8 is dependent on claim 5 and contains both the features and the distinctions over *Kidder* as its parent claim. *Kidder* is not directed to the repair of corruption in a filesystem, but to fault discovery in a computer system and to scaling the level of response according to user need. *Kidder* is concerned with the detection of faults in software and hardware in a system, but does not provide the isolation and repair of a filesystem, as claimed. Instead, *Kidder* relies on external processes to provide needed repair. Thus, the proposed modification when considered as a whole does not result in the invention of claim 8. Accordingly, no *prima facie* obviousness rejection has been presented against claim 8. Therefore, the rejection of claim 8 under 35 U.S.C. §103 has been overcome.

IV. Conclusion

The subject application is patentable over *Kidder* and should now be in condition for allowance. The examiner is invited to call the undersigned at the below-listed telephone number if in the opinion of the examiner such a telephone conference would expedite or aid the prosecution and examination of this application.

DATE: December 26, 2006

Respectfully submitted,

/Theodore D. Fay III/

Theodore D. Fay III
Reg. No. 48,504
Yee & Associates, P.C.
P.O. Box 802333
Dallas, TX 75380
(972) 385-8777
Attorney for Applicants